POCKET COMPUTER NEWSLETTER

© Copyright 1982 — Issue 19 November

PANASONIC ESTABLISHES HHC SERVICE ON COMPUSERVE

Panasonic Company, A Division of Matsushita Electric Corporation of America, One Panasonic Way, Secaucus, NJ 07094, has announced the creation of the Panasonic Users Group (PUG) bulletin board on the nationwide CompuServe network.

The bulletin board service is available to any member of CompuServe. The PUG bulletin board operates as one of a number of Special Interest Groups (SIGs) now available on the network. The service is intended to serve Panasonic's HHC users *as well as users of other hand-held and pocket computers.*

The board already contains a number of programs that can be downloaded directly into HHC units. Of course, a HHC must be equipped with a communications modem in order to connect to the network via telephone.

According to *Edward J. Gelb*, National Sales Manager of HHCs for Panasonic, his company will be providing one hour of free introductory service on the CompuServe network to Panasonic consumer customers who purchase a modem for their HHC. Users can then extend their one hour of introductory time into a regular CompuServe service contract if they are pleased with the world of network information services. (The standard night-time connect charge is $5.00 per hour.)

Current users of the CompuServe network can access the PUG board by using the command GO PCS106 at the command prompt (!) which is given at the end of page CIS-1 after sign-on. Operation of the board is similar to that of other SIG services offered by the network.

One of the primary purposes of PUG is to provide a rapid means of information exchange among HHC and other pocket computer users. This is accomplished through the use of an on-going message bulletin board. Up to 256 messages can exist on this board at one time. Anyone having news of general interest to other users can leave a message on this board. Other members can review current messages and add additional comments, if desired. The medium thus serves as a constant source of up-to-the-minute news on using HHCs, a way to stay in touch with Panasonic software personnel, and share ideas and software accomplishments with other users.

The PUG service may also be used to download programs and documents directly to Panasonic HHCs. Some of the material is also applicable to other brands of hand-held and pocket computers. Current offerings (at no charge, by the way, other than normal CompuServe connect time) include: a program and documentation (operating instructions) for a four-function calculator that can operate in several numeric bases, a program to map memory in a HHC, documentation for operating the new HHC Plotter using BASIC, and a program in BASIC for solving up to 20 simultaneous equations (it appears that this program could readily be adapted to run, say, on a Sharp PC-1500 or Radio Shack PC-2 equipped with an 8K memory module).

The PUG board may also be used to communicate directly with members of the Panasonic HHC staff to obtain prompt responses to user questions. At this time, the PUG SYSOP (system operator) appears to be none other than Mr. Gelb, National Sales Manager for HHCs. That should guarantee pretty prompt responses to any Panasonic HHC users who are in need of assistance.

ROUTINE CONVERTS STATEMENTS

Sharp PC-1500 and Radio Shack PC-2 users will appreciate the use and operation of this small routine. It will automatically scan a program in memory and convert all PRINT statements to LPRINT or vice versa.

An understanding of its operation will enable readers to apply the principles used to other conversion tasks.

Essentially, what the program does is search all of user memory for the exact two-byte code representing the token for a PRINT or an LPRINT statement (depending on which option has been selected). If found, it replaces the second byte with the desired alternate code.

The address at which to begin the search (the start of user program storage) is obtained by examining the contents of memory locations &7865 and &7866. (Remember, the prefix '&' means the number is in the hexadecimal base.) The end of user program storage is available through the STATUS 2 function. These values are used to establish the boundaries of memory that are to be searched. This area is then examined on a byte-by-byte basis for an exact match with the two-byte token being sought. If a match is found (say, with the token for PRINT) then the second byte of the token is changed (to, for instance, represent the LPRINT operation).

To use the program, just load it into memory and type RUN. Respond appropriately to the initial query. (Enter 1 if you want all occurrances of LPRINT to be changed to PRINT, enter 2 if you want the opposite conversion.) Allow the program to run until it signals it has completed the task by beeping. You may then list or CSAVE the converted program. Note that this program is particularly valuable if you developed a lot of programs before you obtained a CE-150 printer/plotter and want to update your library to make use of your new equipment.

This routine was submitted by: *Thomas S. Cox, %Platt Saco Lowell, P.O. Drawer 2327, Greenville, SC 29602.*

Program *Tokens Converter*

```
1:"LP"INPUT "LPT
  -PRT(1) OR P-L
  PT(2)";A:IF A=
  1GOTO 7
2:IF A=2GOTO 4
3:END
4:C=PEEK &7865*2
  56+PEEK &7866:
  FOR I=CTO (
  STATUS 2-1):A=
  PEEK I:B=PEEK
  (I+1):IF (A=24
  0)+(B=151)=2
  GOSUB 6
5:NEXT I:BEEP 20
  :END
6:POKE (I+1),185
  :RETURN
7:C=PEEK &7865*2
  56+PEEK &7866:
  FOR I=CTO (
  STATUS 2-1):A=
  PEEK I:B=PEEK
  (I+1):IF (A=24
  0)+(B=185)=2
  GOSUB 9
8:NEXT I:BEEP 20
  :END
9:POKE (I+1),151
  :RETURN
```

UNDERSTANDING THE SHARP PC-1500

This is the third article in a series being presented by: *Norlin Rober, 407 North 1st Avenue, Marshalltown, IA 50158.*

More Hidden Instructions

In Issues 15 and 16 of *PCN*, the uses of PEEK and POKE were discussed. Here are a number of other instructions of which you may not be aware.

The PC-1500's processor can address two separate banks of memory, each of which could conceivably contain as much memory as 64K. The first bank addresses the System ROM, CE-150 (Printer/Cassette) ROM, System RAM and user RAM. At present, at least, the "alternate buffer" (mentioned in Radio Shack's Instruction Manual, but not Sharp's) contains only the input/output (I/O) ports. The PEEK# and POKE# instructions operate in the same way as PEEK and POKE, except that they address the memory in this second bank of memory.

The use of POKE# can affect the buzzer, clock, remote switches that control the cassette recorders, pen solenoid, paper feed, pen position, and other I/O functions. Be forewarned that using POKE# can lead to serious problems – not only system crashes, *but possibly physical damage to connected devices.* I'll give you an example: While experimenting with POKE#, I had turned on the pen solenoid, but had not realized that it remained on. A few minutes later I somehow happened to notice that the solenoid had become quite hot! As it turned out, no permanent damage had been done, but luck must have been with me.

Thus, unless you know what you are doing (or you like to live dangerously!), you would be well advised not to experiment with the POKE# instruction.

The use of CALL will cause execution of a machine-language program beginning at a specified memory address. Those who choose not to delve into machine language can still use CALL in certain ways, provided the desired address to call is known. I'll give you an example that you might find interesting and useful.

The power-up routine checks certain flags to determine whether the cause of the last power-down was the automatic 7-minute shutoff. If the automatic shutoff *had* occurred, certain parts of the power-up routine are skipped over. The operations skipped include clearing the display, canceling execution of a program not finished, re-seeding the random number generator, and running the pen through its color change cycle.

You can perform the equivalent of a 7-minute shutoff (without having to leave the computer on for seven minutes!) by executing the instruction CALL &E33F. This instruction may be used as part of a BASIC program too, in case you want the computer to turn itself off under program control. *[It is a nifty command to assign to a softkey in the Reserve mode, such as: F1: CALL &E33F@. Remember, the "@" gives you immediate execution of a softkey! – N.W.]* It is also possible to obtain the equivalent of having used the OFF key, by execution of CALL &CD71.

It is possible to clear the variables in "main" memory (without clearing those in "fixed" memory, that is A – Z and A$ – Z$), by execution of CALL &D091. This would permit re-dimensioning of variables without loss of the fixed variables, within a program.

Another instruction not mentioned in the manuals is the NEW command followed by an expression. Nor is the difference between just plain NEW and NEW0 made clear.

NEW will effectively clear a program (by setting the End Program pointer at &7867-68 to coincide with the Start Program pointer at &7865-66) without changing the starting location. On the other hand, NEW0 resets both pointers to &C5 plus the beginning address of RAM physically present in the computer. Another difference is that NEW0 will cancel LOCK mode, whereas NEW will not.

When NEW is followed by an expression (within the permissable range), the value of that expression determines the memory location at which program lines will be entered. Thus, execution of the directive NEW &4800 will take address &4800 as the beginning location at which program lines will be placed into memory, whether entered manually (in PRO mode) or loaded from cassette. Addresses below &4800 are then protected.

Machine-Language Programmers Take Note!

One use for this NEW <expression> command is to reserve space for a machine-language program in such a way as to protect it from being changed by storage of data or BASIC program lines.

Another use is reserving a segment of memory for the efficient storage of data with PEEK and POKE. (Such data would need to consist of integers from 0 to 255 or, with appropriate programming, one could store larger integers.)

It is not possible, however, to execute NEW &4000 (NEW &3800 if an 8K module is used) to absorb Reserve memory into program space. A safeguard is built into the NEW routine to prevent this. Attempting to do so results in the appearance of ERROR 25. That same message appears if you attempt starting your program at an address higher than that of existing memory (e.g., NEW &6800).

If the Start of Program pointer has been relocated by, for example, NEW &4800, executing NEW will leave it there (while effectively clearing the program, of course); but, NEW0 will set it back to its normal location.

Special Cassette Commands

A considerable amount of versatility is added to the cassette storage system by the use of the CLOAD M and CSAVE M instructions. Execution of CSAVE M permits saving memory contents between two specified memory addresses. CLOAD M reloads what has been stored on a tape by CSAVE M. Machine-language users will immediately recognize this as a very convenient method for saving machine-language routines.

To illustrate: Suppose you have an 8K expansion RAM module. You have placed a machine-language program into the area &38C5 to &39D0, having left Reserve memory (&3808-C4) to operate in the normal manner. Suppose further, that you have executed NEW &3A00 to separate your BASIC program from the portion reserved for the machine-language program. You can now save the machine-language program by executing CSAVE M "NAME"; &38C5, &39D0. (If you would like to save Reserve memory along with it, then just use the directive CSAVE M "NAME"; &3808, &39D0.) The BASIC portion can be CSAVEd in the usual manner.

When you want to reload this program at some later time, you execute CLOAD M "NAME" or simply CLOAD M. This will place the machine-language portion back into its original location in the computer's memory. Of course, NEW &3A00 must be executed before CLOADing the BASIC program or else the machine-language routine would be obliterated.

CLOAD M "NAME"; <address> is also a valid command. Its effect is to load the contents of the cassette into memory beginning at the address given, rather than at the same location it had when CSAVEd.

If you are using two cassette recorders, you may find it of interest to note that CLOAD M-1 and CSAVE M-1 are valid instructions as well!

Unfortunately, there appears to be no way to verify a recording made using CSAVE M. As you might guess, there are no MERGE M or CHAIN M instructions.

The idea might occur to you that we have here a way to save both Reserve memory contents and a BASIC program simultaneously with CSAVE M &3800, STATUS 2. Later CLOAD M might be used to load both back into the computer. But, there is a hitch! It does get loaded back into the computer all right, but the End of Program pointer does not get set. Thus, the computer is mislead into thinking that it doesn't contain a program at all.

CLOAD M will load only a recording made by a CSAVE M instruction. A recording made with CSAVE M can be CLOADed by only the CLOAD M command.

[That is some super information. Thanks, Norlin. If anyone is looking for a good project to apply their machine-language programming skills on, take a look at the next article. With all of the m.l. material that has been supplied through the pages of PCN, *I expect to see lots of clever m.l. and hybrid programs appearing on the scene in the near future! – N.W.]*

RENUMBERING PROGRAM REVISITED

It soon became apparent from the mail received after Issue 18 of *PCN*, that many readers could use an explanation of the renumbering program provided by *Milt Sherwin, 8602D East Amherst Drive, Denver, CO 80231.* Here is how Milt describes its development and operation:

Program Concepts and Development

In developing a renumbering program, it is necessary to know the format of the BASIC lines (line number storage, linkage arrangements, BASIC text representation, etc.). Once this information is obtained, the choices revolve around how sophisticated the program should be, e.g. (1) how much flexibility will the user be provided in choosing the parameters, (2) does the program renumber line numbers following GOTOs, GOSUBs, THENs, and REMs, and (3) how easy the program is to use in general. By far the most difficult item is (2), the renumbering of GOTOs, GOSUBs, THENs and REMs. Of course, this directly influences item (3), how easy the renumbering program is to use!

BASIC program storage begins at hexadecimal address &38C5 (which is decimal 14533) in a PC-1500 equipped with an 8K model CE-155 RAM module. (The address is &40C5 if there is no memory module or only a 4K module is installed.)

BASIC lines are stored in memory with the following format:

1 byte	1 byte	1 byte	N bytes	1 byte	1 byte
Line Number (high)	Line Number (low)	Pointer	BASIC text	Carriage Return "&0D"	*

The byte denoted by an asterisk here contains the high byte of the next line number or &FF if this is the last line of the program. This works OK since the highest line number allowed by the interpreter is 65279 (hexadecimal $FEFF).

Line numbers at the beginning of statement lines are stored as two bytes (high, low). Thus, line 10 decimal equals bytes 00 0A in hexadecimal. Likewise, 65279 converts to FE FF.

The value stored in the byte marked "pointer" indicates how many bytes to the line-ending carriage return (taken from the pointer location itself).

BASIC lines are made up of BASIC tokens and ASCII coded characters. The code for carriage return indicates the end of a BASIC line. Thus, the line "300 GOTO 90" would be stored in the PC-1500 as:

01 2C 05 F1 92 39 30 0D

Knowing this format enables a programmer to "step" through a program, line-by-line, using PEEKs and a few program loops.

Once the program starting point and the format of BASIC lines are known, it is easy to find the line number locations and appropriate tokens (for GOTO, GOSUB, THEN or REM) that would be associated with renumbering. It is also possible to determine the line numbers that follow those tokens, using the string-manipulating functions CHR$, LEN and VAL, etc. However, the line numbers following tokens are stored as ASCII digits, and line numbers such as 5, 50, 500 and 5000 *all require a different number of bytes for storage* (1, 2, 3 and 4 respectively). A comprehensive renumbering program must take this into consideration. The plot, thus, thickens! If a program to be renumbered contains a line such as "300 GOTO 90" and renumbering causes it to become "310 GOTO 100", then an additional byte is needed because the number following the GOTO statement has now expanded to three digits! That means everything stored after that location (perhaps an entire program!) must be relocated by one byte to create room for the new digit. This is not an easy undertaking. A similar problem occurs in the opposite direction when a referenced number decreases, such as when "310 GOTO 100" becomes "300 GOTO 90". Now one byte must be removed and everything above that point moved "down" appropriately. One more little point: If the line length changes because digit(s) are added/subtracted, then the line length "pointer" must also be altered to reflect the new line length!

Note that these problems do not occur with statement line numbers. They are always stored in hexadecimal, two-byte notation. Hence, statement line numbers are easily modified without encountering expansion/contraction problems.

There is another aspect of program renumbering which must be considered during program design. The relationship of the original line numbers to the renumbered line numbers must be kept track of in order to renumber the line numbers following the GOTOs, GOSUBs, THENs and REMs. The location of this line number relationship table could be critical to the program's operation. During the operation of the renumbering program, the variables storage area could be disturbed if the program size changes due to the altering of line numbers following tokens. One way to avoid this potential problem is to POKE (store) and PEEK (read) numbers directly into "safe" areas in memory. I selected two areas which should be OK for this purpose. Namely, the fixed variable storage areas for character variables E$ through O$ (at addresses &7050 through &70FF) and P$ through Z$ (at addresses &7150 through &71FF). None of these character variables are used by the main part of the renumbering program. *[And, it makes no difference if these areas are used by the program being renumbered. After all, you cannot execute the program being renumbered at the same time that the renumbering is occurring! – N.W.]* Thus, this area is effectively protected *whenever it is being used for storage of the number relationship table.*

The first area, &7050 through &70FF is used to hold the original line numbers. The 176 bytes here can store 88 line numbers. The second area, &7150 through &71FF can also hold 88 line numbers too. So, that is where the renumbered line values are held.

If you have more than 88 lines in a program, then renumber it in sections! *[Or, use the information provided by Norlin Rober in this issue to relocate the line number relationship tables to a larger, protected area of memory. After all, now we know how to set aside as much memory as we want for special purposes! – N.W.]*

Next, there is the problem of "where does the renumbering program itself reside in memory?" If it is made a part of the main program (numbered so that it resides at the end of the program being developed), then the dynamics of the changing line lengths in the program being renumbered will effect the renumbering program as it tries to operate. That won't work.

A separate renumbering program could be written which could be MERGEd with the program under development. Would this approach work? The MERGE command is indeed interesting. It allows a user to have more than one program in memory at one time. If each program uses alphabetical labels, it can be accessed and executed. However, only the last program MERGEd can actually be modified by the system's editor. Furthermore, no selectivity is provided if a program is LLISTed or CSAVEd. Everything in memory is simply listed or saved. Furthermore, in order to develop a new program one must initially MERGE a program from tape. You cannot just start typing in line numbers and BASIC statements. And, even if the renumbering program is MERGEd after program development and is used to renumber the program already in memory, there is still a problem. While the renumbering program could then be removed line-by-line, the program under development would still not be accessible to the editor. This is because the &FF byte which terminated the MERGEd renumbering program would still be in memory following the &FF byte that terminated the original program under development. That is, there would be two consecutive &FF bytes in memory which would prohibit alterations being made by the editor. Of course, the program could be CSAVEd and then CLOADed. But, that takes a lot of time and is, frankly, cumbersome. But, is there light at the end of the MERGE tunnel?

What if the renumbering program were loaded first? Then, development work could take place in memory beyond the renumbering program. The program under development could also be expanded or contracted without disturbing the renumbering program. The renumbering program would not have to be MERGEd for each use or taken out to continue the development process. But, what about overcoming the previously mentioned problems with MERGE?

There happen to be three pointers (taking a total of six bytes, each has a high and low part) which indicate the memory address of what I call the "start of BASIC" (SOB), the "end of BASIC" (EOB) and the

"start of program" (SOP) locations.

30821	30822	30823	30824	30825	30826
Start of BASIC (SOB)		End of BASIC (EOB)		Start of Program (SOP)	
High	Low	High	Low	High	Low
&7865	&7866	&7867	&7868	&7869	&786A

In the non-MERGE mode of operation, pointers SOB and SOP are the same. When there is a MERGE, pointer SOP contains the starting address of the MERGEd program. This is the reason that additions, changes and deletions only work on the last program MERGEd. Pointer EOB points to the &FF byte of the last program in memory. A &FF byte also terminates every program that is MERGEd into memory. The SOB pointer normally remains fixed at &38C5 (14533 decimal) in a PC-1500 containing an 8K memory module. Would it be possible for a renumbering program to modify these pointers?

Yes indeed! And, that is just what my renumbering program does. The methodology is as follows:

1) The renumbering program is CLOADed into memory.

2) A RUN command causes the renumbering program to alter the SOP and EOP to simulate the performance of a MERGE.

3) Program development may then take place. The program being developed should be given a label for use when testing. (Thus, one can say LIST "A" or RUN "A" as appropriate.)

4) Subsequent RUN directives cause the renumbering program to renumber the program that is under development.

5) At the end of a renumbering pass. the renumbering program asks whether it should be "unlinked." If so, the program modifies SOB so that it equals SOP, thereby effectively removing the renumbering program from further access.

6) Once the renumbering program has been unlinked, the program under development can be LISTed, CSAVEd or executed using normal methods.

7) Execution of the command NEW0 restores all three pointers (SOB, EOB and SOP) to their pristine states.

Furthermore, if a previously developed program needs to be renumbered, it can simply be MERGEd in place of step 2 above. Doing so causes SOP and EOB to be set appropriately by the MERGE operation instead of by the renumbering program.

For reference purposes, the behavior of the SOB, EOB and SOP pointers are summarized here:

DURING REGULAR OPERATIONS	
NEW0/START	SOB = EOB = SOP = 14533
PROGRAMMING/CLOAD	SOB = SOP = 14533, EOB > 14533
MERGE	SOB = 14533, SOP > 14533, EOB > SOP
DURING RENUMBERING OPERATIONS	
NEW0/START	SOB = EOB = SOP = 14533
CLOAD RENUM	SOB = SOP, EOB > 14533
INITIAL RUN/MERGE	SOB = 14533, SOP = EOB > SOB
DEVELOPMENT	SOB = 14533, SOP > SOB, EOB > SOP
UNLINK	SOB = SOP(Development), EOB > SOP
NEW0	SOB = EOB = SOP = 14533

(Value of 14533 is for a PC-1500 with an 8K memory module installed.)

Operation of the Renumbering Program

Now that you are familiar with the concepts behind the design of the program, the actual implementation of the program can be described. Please remember that the program only renumbers statement line numbers and line numbers referenced by GOTO, GOSUB, THEN and REM statements. *It will not recalculate line numbers that are obtained by manipulating variable names or performing similar types of programming tricks!*

Lines 100 – 140: The values of SOB and SOP are compared to determine if they are equal, indicating that a MERGE has not been performed. If equal, a MERGE is simulated, program execution terminates and the user can undertake program development. If SOB and SOP are not equal, then the program assumes that program development has taken place (either through an actual tape MERGE or a previous execution of this program).

Lines 150 – 250: Variables are initialized and renumbering parameters are requested from the operator. There are four parameters used by the program:

PARAMETER	DEFINITION	DEFAULT VALUE
ST	Starting line number	First line of the development program
IC	Line increment	10
EN	Ending line number	Last line of the development program
LN	Initial line number when program renumbered	100

Commas must be used to separate the parameters during input *even if values are not entered for some parameters.* Thus, inputting ",,,200" will result in the current development program being renumbered in its entirety, with line increments of 10, and the first line of the renumbered program being 200. However, if the RETURN key is pressed without entering any parameter values or commas, then default values will be used to renumber the program. The parameters are input as a single character variable, then split into four individual variables for use within the program.

Lines 260 – 300: The variables used to store the parameters are set for operation at their default values or the values inputted by the user.

Lines 310 – 360: The starting location is found.

Lines 370 – 450: Line numbers are renumbered. The original line number/renumbered line number table is built. The subroutine at line 530 is used to determine the original line numbers.

Lines 460 – 480: Variables are set for renumbering line numbers that follow GOTOs, GOSUBs, THENs and REMs. Each line is searched for the corresponding tokens using the subroutine at line 570.

Lines 490 – 520: The operator is provided the option of unlinking the renumbering program. If unlinking is to occur, SOB is set equal to SOP. If not, the program is terminated. Further program development may then take place.

Lines 530 – 560: Subroutine to determine a line number. Line numbers, stored as two bytes, are converted to single line number values.

Lines 570 – 610: First part of the subroutine that searches for GOTO, GOSUE, THEN and REM tokens. (Renumbering line numbers after a REM only works if the line number follows immediately after the REM.) If no appropriate token is found, detection of the code for carriage return in line 590 causes the subroutine to terminate (via line 780).

Lines 620 – 650: Second part of the token-searching subroutine. If an appropriate token is found, the ASCII representation of the associated line number(s) is(are) located. As they are found, they are converted so they may be stored in a character variable for later use.

Lines 660 – 710: Part three of the subroutine evaluates the line number stored as a character variable. The previously created line number table is used to determine the referenced renumbered line number.

Lines 720 – 780: The fourth part tests to determine if the renumbered line number has the same number of digits as the original line number. If the renumbered line number has more digits than the original line number, the subroutine starting at line 790 is used to create more space. If the new number has less digits, the subroutine at line 830 is used to remove the excess space. If the line numbers have the same number of digits, the renumbered line number is placed in position by the subroutine at line 860. If no match is found in the table, no renumbered line number is required (as the original line number has not changed). The variables are then reset. The search for more line numbers or tokens resumes.

Lines 790 – 820: Subroutine to create additional space in a program.

Lines 830 – 870: Subroutine to remove excess space in a program.

Lines 880 – 910: Subroutine to place a renumbered line number into memory.

[OK, all you machine language programmers. Here is an opportunity for you to really have some fun. Milt has described in detail how his renumbering program operates. Machine-language speed would be nice in this application. Who is going to tackle the project? – N.W.]

MEMO PRINTER PROGRAM

Here is a program for Radio Shack PC-2 and Sharp PC-1500 users who want to keep notes or text in memory. This program was created by: *Brian Peterson, 6807 N. Sheridan Road, Chicago, IL 60626.* One of the interesting features about this program is that *it prints sideways!* Thus, it accepts and prints out full 80-character lines. Brian calls his program "Memo Printer." Here is what he has to say about using it:

It Operates In Three Modes

These modes are referred to as: Input, Edit and Print.

The Input mode allows the user to enter characters into memory. You need at least a 4K memory expansion module to use the program. With a 4K module, it can accept approximately 33 lines of text. An 8K module allows about 88 lines. Theoretically, a 16K expansion unit would allow up to about 187 lines. Each line may contain up to 80 characters.

If you want to leave a blank line between paragraphs, just enter a single 'space' character on the line. To get out of the Input mode, press the ENTER key without putting any characters in the line.

Editing

You use the Edit mode to go back and change text that was originally entered while in the Input mode. The Edit mode allows you to examine the text buffer and make alterations. You can insert characters and entire lines, if desired.

Several keys have different functions than normal when in the Edit mode. The DEF key is used to activate the ability to insert or delete lines. After the DEF key is pressed, the PC will ask whether the user wishes to insert or delete. Enter 'I' to insert. Enter 'D' to delete. When in the delete phase, the CL key deletes individual characters. Holding the key down will cause it to repeat and delete a series of characters. The MODE key enables the user to insert characters at the current cursor position. When this insert function is in effect, the symbol > will appear in its inverse form (filled-in) on the screen. The key normally used to select the desired Reserve mode (◊) is used to append lines to the end of the text. The scroll line up (^) key and the scroll line down (v) key are used to examine lines above or below the current text line. The symbols <, >, ?, :, ; and comma are available through the use of the SHIFT key. Use the SML key to enter lower case characters. Tip: When entering text, put the unit in the lower case mode and use the SHIFT key when a capital letter is needed.

The six softkeys can be customized to allow the use of characters not usually available on the keyboard. Program lines 247 -- 252 can be altered for this purpose.

Press the keys firmly when using the Edit mode to allow time for the character to be recognized. Most keys will repeat if held down in this mode.

To exit the Edit mode, press the RCL key.

Printing

This program prints "lengthwise" on the paper. Options for selecting the pen color as well as single- or double-spacing are provided.

Use GOTO

Once you have text in memory, always enter the program using a GOTO directive, not RUN. Using RUN will clear out any text that you have placed in memory.

There Is No Stopping

This program disables the ON/BREAK key during operation, except when printing.

[While this last feature is interesting, I am not convinced that it is desirable. I was recently using the program while away from the office, got into the Edit mode, and forgot how to get out. With the BREAK key disabled, I had no choice but to leave the unit on until I got back to the office, as I did not want to reset the PC and lose everything that I had in the unit. I don't think the risk of accidently striking this key, nor the possible consequences of doing so, are worth not being able to shut the unit off if I forget how to operate the program. In any event, a word to the wise: Keep the operating instructions handy until you have memorized all of the options available!— N.W.]

Program *Memo Printer (Part 1)*

```
 5:ARUN :WAIT 0:
   POKE# &F00D,
   PEEK# &F00DOR
   128
10:IF STATUS 3=&5
   800DIM P1$(0)*
   80,P2$(0)*80,A
   $(32)*80:N=33
11:IF STATUS 3=&6
   000DIM P1$(0)*
   80,P2$(0)*80,A
   $(87)*80:N=88
12:IF STATUS 3=&6
   400DIM P1$(0)*
   80,P2$(0)*80,A
   $(186)*80:N=18
   7
13:POKE STATUS 3-
   23,72,118,74,0
   ,5,189,255,65,
   78,78,153,8
14:POKE STATUS 3-
   11,76,119,139,
   6,72,119,74,0,
   158,18,154
15:PRINT "Input
     Edit   Print
     End"
16:V=ASC INKEY$ :
   IF VCALL
   STATUS 3-23
17:IF VAND ASC
   INKEY$ THEN 17
18:ON V-16GOTO 20
   ,16,200,16,40,
   600
19:GOTO 15
20:LOCK :L=0:
   USING
25:ON ERROR GOTO
   0
30:PRINT "Line";L
   ;">";:INPUT ""
   ;A$(L):CLS :L=
   L+1:IF L<NGOTO
   30
35:M=L-1:UNLOCK :
   BEEP 1:CLS :
   GOTO 15
40:PRINT " BLK BL
   U GRN RED  -CO
   LOR-"
41:V=ASC INKEY$ :
   IF VCALL
   STATUS 3-23
42:IF VAND ASC
   INKEY$ THEN 42
43:C=V-17:IF V<17
   OR V>20GOTO 40
44:PRINT " SNG
     DBL     -SPA
   CING-"
45:V=ASC INKEY$ :
   IF VCALL
   STATUS 3-23
46:IF VAND ASC
   INKEY$ THEN 46
47:IF V<>17AND V<
   >19GOTO 44
48:S=(V=17)+2*(V=
   19):T=11*(V=17
   )+6*(V=19)
49:CLS :GRAPH :
   LINE -(220,0),
   0,C:TEXT :LF 5
50:GRAPH :SORGN :
   ROTATE 1:CSIZE
   2
52:POKE# &F00D,
   PEEK# &F00DAND
   127
55:G=0
60:FOR X=1TO 61
   STEP 20
70:FOR Y=T-1TO 0
   STEP -1
75:X$=MID$ (A$(Y+
   G),X,10)
80:IF X$=""GOTO 1
   10
90:GLCURSOR (200-
   19*Y*S,-X*12)
100:LPRINT X$
110:NEXT Y
120:FOR Y=0TO T-1
125:X$=MID$ (A$(Y+
   G),X+10,10)
130:IF X$=""GOTO 1
   60
140:GLCURSOR (200-
   19*Y*S,-(X+10)
   *12)
150:LPRINT X$
160:NEXT Y
170:NEXT X
```

program continued on next page

Program *Memo Printer (Part 2)*

```
175:G=G+T:IF M>=G
    TEXT :LF 5:
    GRAPH :SORGN :
    ROTATE 1:GOTO
    60
180:TEXT :LF 5:
    BEEP 1:POKE# &
    F00D,PEEK# &F0
    0DOR 128:GOTO
    15
200:ON ERROR GOTO
    350:L=0
210:CLS :WAIT 0:P=
    1:Q=1:R=5:IF D
    LET L=L-(L>0)
220:USING "####":
    GOSUB 580
221:Z=0:E=0:F=0:
    GOTO 224
223:Q=1:R=P+4:IF P
    >21LET Q=P-20:
    R=25
224:PRINT L;">";
    MID$ (A$(L),Q,
    21):IF ZCURSOR
    4:GPRINT 0;127
    ;62;28
225:Y=0:CURSOR R:
    PRINT CHR$ 127
226:K$=INKEY$ :V=
    ASC K$:IF V>32
    GOTO 270
227:IF VGOTO 230+V
228:X=X+1:IF X<6
    GOTO 226
229:X=0:Y=(Y=0):IF
    YCURSOR R:
    PRINT CHR$ 127
    :GOTO 226
230:CURSOR R:PRINT
    MID$ (A$(L),P,
    1):GOTO 226
231:W=2:GOSUB 550:
    E=(2=(2AND
    PEEK &764E)):
    GOTO 225
232:W=8:GOSUB 550:
    F=(8=(8AND
    PEEK &764E)):
    GOTO 225
238:P=P-(P<>1):
    GOTO 223
239:M=M+(M<N-1):L=
    M:GOSUB 500
240:L=L+(L<M):P=1:
    Q=1:R=5:GOTO 2
    21
241:L=L-(L<>0):P=1
    :Q=1:R=5:GOTO
    221
242:P=P+(P<=LEN A$
    (L)):GOTO 223
243:P=1:Q=1:R=5:
    GOSUB 580:
    GOSUB 500:GOTO
    221
247:K$="!":GOTO 27
    2
248:K$=",":GOTO 27
    2
249:K$="#":GOTO 27
    2
250:K$="$":GOTO 27
    2
251:K$="%":GOTO 27
    2
252:K$="&":GOTO 27
    2
254:P1$(0)=LEFT$ (
    A$(L),P-1):P2$
    (0)=MID$ (A$(L
    ),P+1,80-P):A$
    (L)=P1$(0)+P2$
    (0):GOTO 223
255:GOSUB 500:
    GOSUB 580:L=M:
    GOTO 15
257:W=128:GOSUB 55
    0:CLS :GOTO 40
    0
261:Z=(Z=0):GOSUB
    500:GOTO 224
262:GOTO 272
270:IF E+F=1IF V>6
    4LET K$=CHR$ (
    VOR 32)
271:IF V>39IF V<48
    IF E=1GOSUB V-
    39+600
272:POKE &764E,253
    AND PEEK &764E
    :E=0
273:IF P>80GOTO 35
    0
274:IF Z=0POKE (
    STATUS 3+6+P+8
    0*L),ASC K$:P=
    P+1:GOTO 223
275:P1$(0)=LEFT$ (
    A$(L),P-1)
280:P2$(0)=MID$ (A
    $(L),P+(Z=0),8
    0-P)
290:A$(L)=P1$(0)+K
    $+P2$(0)
300:P=P+1:GOTO 223
350:BEEP 1:GOTO 22
    5
400:PRINT "Insert
    or Delete?":
    POKE &764E,127
    AND PEEK &764E
401:V=ASC INKEY$ :
    IF VCALL
    STATUS 3-23
402:IF VAND ASC
    INKEY$ THEN 40
    2
403:IF V<>68AND V<
    >73THEN 400
410:IF V=68INPUT "
    Delete line#";
    D:FOR X=DTO M:
    A$(X)=A$(X+1):
    NEXT X:A$(M)="
    ":M=M-1:L=D:
    GOTO 210
420:INPUT "Insert
    line after lin
    e#";I:I=I+1
430:FOR X=M-(M=N-1
    )TO ISTEP -1:A
    $(X+1)=A$(X):
    NEXT X:A$(I)="
    ":M=M+1:L=I+1:
    GOTO 210
500:IF ASC INKEY$
    THEN 500
510:RETURN
550:U=PEEK &764E:
    IF UAND WPOKE
    &764E,U-W:
    GOSUB 500:
    RETURN
555:POKE &764E,U+W
    :GOSUB 500:
    RETURN
580:POKE &764E,245
    AND PEEK &764E
    :E=0:F=0:
    RETURN
600:POKE# &F00D,
    PEEK# &F00DAND
    127:END
601:K$="<":RETURN
602:K$=">":RETURN
603:K$=":":RETURN
604:K$=";":RETURN
606:K$=",":RETURN
608:K$="?":RETURN

STATUS 1
                3376
```

STOPWATCH PROGRAM FOR R. S. PC-1/SHARP PC-1211

Radio Shack PC-1 and Sharp PC-1211 users can use this program to precisely time events down to 1/10th of a second intervals!

Timer Design

Two conditions (rules) should be met when designing an accurate time keeping device: 1) There should be a large number of cycles ("ticks") per second, and 2) The regularity of each cycle must be as stable as possible. Both of these rules are met quite nicely by the Cesium Atomic Clock. It "ticks" at a frequency of 9,192,631,770 Hertz (cycles per second). Its regularity is within 1 part in 10,000,000,000,000. That makes it the most accurate timing device known to man.

The design rules mentioned, of course, can be applied to the design of computer "clock" programs. Unfortunately, many programmers do not follow these rules. Consequently, inaccurate (errors up to several minutes each hour) times results.

Environmental temperatures can also have an effect on accuracy. Laboratory experiments performed on the Radio Shack PC-1 and the Sharp PC-1211 indicate that program execution speed increases slightly with increasing temperature. In equation form, it was found that:

S(T)/S(T=0) = 2.0261E-5*(T+273.15)+0.994466

when T is between 0 and 40 degrees Centigrade. S(T) is the relative execution speed, T is the temperature in Celsius of the PC, and S(T=0) is the reference speed at 0.0C (defined, arbitrarily, as 1.000) in this equation. Using this equation, one can observe that at 40 degrees C., a program executes about 0.08% faster than the same program at a temperature of zero. Though these effects are relatively small, they can be significant over an extended period of time. The program provided

in this article utilizes a temperature compensator that may be utilized in extreme temperature ranges.

Other experiments indicated that battery condition had little effect on the execution speed. Using very weak batteries in the PC (with the display barely visible) gave the same results as fresh batteries. This is good news. It means you can run the accompanying program for extended periods of time without fearing a loss in accuracy due to battery condition.

The stopwatch program provided here has been pre-calibrated against a laboratory chronometer. Chances are, it will not require any additional calibration in your PC. However, if you should find the program slightly fast or slow on your particular PC, it contains an easy to use calibration routine. This routine calculates the exact calibration constant required for an individual PC. The routine, if used at all, only needs to be run once after loading the program into memory.

A Stopwatch Program

This program makes a good substitute for an ordinary 1/10th second stopwatch. It can be calibrated to give errors of less than about 0.2 seconds per day. While this is not quartz accuracy, it is better than some mechanical spring-wound watches. The output is given in hours, minutes, seconds and tenths of a second.

The design rules discussed above were used in designing this program. For instance, examine the basic time-keeping routine in line 8 of the program.

To satisfy the first rule, this line is made to execute as fast as possible by making it short and simple as well as using a high speed variable (Z).

The second rule (perhaps more important), is satisfied by assigning Z a large initial value (see line 4). This was done because tests have shown that a calculation of Z=Z+1 takes slightly longer as the order of magnitude of Z increases. Thus, it would not be wise to start off with Z=0, as the "clock" would "tick" slower and slower as Z rapidly increased several orders of magnitude. Instead, an initial value of Z=800000 is used. It then takes several days before the order of magnitude changes and each "tick" is thereby made constant.

Operation

The program is designed to be executed in the DEF mode. You press SHIFT/S to place the PC stopwatch into the start position. Then, to actually start timing an event, just press the ENTER key. Use the BREAK key as the stop button. To display the elapsed time, simply press SHIFT/D. An example output might appear as:

TIME (H.MMSST) = 2.07583

This would be interpreted as 2 hours, 7 minutes and 58.3 seconds.

Calibration

If you should find the stopwatch program to be slightly off on your particular unit, you can execute the calibration routine. To obtain a calibration you need to first run the program, as described above, then press SHIFT/C. An example illustrates the process:

You time an interval of exactly 4 hours, 0 minutes and 0 seconds. But, when you press SHIFT/D to display the elapsed time, you find that the stopwatch is off by a significant amount. So, you run the calibration routine:

SHIFT/C
CALIBRATION - - >
TRUE (H.MMSST): 4.0000
NEW Y = 0.17658008

You would have entered the value '4.0000' in the above example. The NEW Y value would then have been computed by the calibration routine. The calculation uses the difference between the actual elapsed time and that originally obtained using the stopwatch program.

You complete the calibration process by placing the PC in the PRO mode, then changing the constant Y in lines 4 and 32 to the new value. You then use this value in the future on your PC.

Note that a calibration run should be made over a several hour period so as to keep relative errors to a minimum. Also, you need run this procedure only once (if at all) for a particular PC.

If you are using your PC at temperature extremes that are greater than 85 degrees Fahrenheit or less than 55 F., then you might also want to run the temperature calibration routine. This routine assumes you have already calibrated your PC for normal temperatures. Press SHIFT/X to perform this calibration and respond to the prompt for temperature and the scale. The routine will provide a new value of Y for you to use in lines 4 and 32.

Summary

The program is designed to be used in the DEF mode. Here is a list of the DEFined keys and their functions:

SHIFT/S – Puts the PC into the 'ready to start' position.
ENTER – Starts the timing of an event.
BREAK – Stops the timing of an event.
SHIFT/D – Displays the elapsed time.
SHIFT/C – General calibration procedure.
SHIFT/X – Extreme temperatures calibration procedure.

This program provided by: *Emerich Auersbacher, 41 King Street #2, Belleville, NJ 07109.*

Program *Stopwatch*

```
 1 BEEP 1:PAUSE "   STOPWATCH PROGRAM"
 2 PRINT "USE (DEF) MODE PLEASE.":END
 4 "S":Z=800000:Y=0.1766080237:BEEP 1
 6 PRINT "READY: (ENTER) TO START"
 8 Z=Z+1:GOTO 8
10 "D":T=DMS((Z-800000)*Y/3600):T=T+0.000005
12 PRINT USING "###.#####";"TIME(H.MMSST)=";T:
   GOTO 4
14 "C":PAUSE "CALIBRATION -->"
16 INPUT "TRUE (H.MMSST):";T:T=DEG T
18 T=3600T:Y=T/(Z-800000)
20 PRINT USING;"NEW Y =  ";Y:END
22 "X":PAUSE "TEMP. ADJUST -->":INPUT "TEMPERA
   TURE:";T
24 INPUT "DEGREES C OR F?";D$
26 IF D$="F" LET T=5/9*(T-32):GOTO 32
28 IF D$="C" GOTO 32
30 GOTO 24
32 Y=0.1766080237:Y=1.000416768*Y/(0.000020261*
   (T+273.15)+0.994466)
34 PRINT USING;"NEW Y=  ";Y:END
```

A FEW OP-AIDS FOR THE PC-1500/PC-2

A fellow by the name of *Rich Hart, 2019 N. Kessler, Wichita, KS 67203*, recently sent in a few routines to share with *PCN* readers. They come in pretty handy, especially if you are inclined towards "exploring" the Sharp PC-1500 or Radio Shack PC-2.

The first routine, shown in lines 1 – 3 of the accompanying listing, just combines the use of the STATUS 1 and STATUS 2 commands to automatically display the total number of bytes used for program storage. It uses the ARUN command to display this information each time the PC is turned on.

The first routine then calls part of the next routine to obtain the address range being utilized for program storage. This second routine comprises lines 10 – 50 of the accompanying listing. It converts the value in variable N to hexadecimal notation. It is particularly handy when examining the contents of memory locations using the PEEK instruction. Using the DEF J keystroke sequence, decimal values on the display are immediately converted to their hexadecimal equivalents. Note that this routine can be used as a subroutine by entering

it at line 12. However, when this is done, you must set up variable SF=1 as a "flag" so that the latter part of line 45 will be executed.

The third routine finds the absolute hexadecimal address of the first byte of code for a given BASIC program line number. It is useful in a number of situations, such as when you want to perform an ASCII dump of just a portion of a large BASIC program. Those of you puttering around with machine language or considering hybrid program (part BASIC, part machine code) will also find this utility quite useful. This routine occupies lines 200 – 230 in the listing.

It operates as follows: After inputting the statement line number that is to be located, the user presses DEF/K. Line 220 of the routine examines the first two bytes of a BASIC source line (which holds the line number) and converts it from its two-byte hexadecimal format so that it can be compared with the decimal value specified. If a match is found, then the current address is displayed. Otherwise, the routine takes the third byte as a displacement value to get to the start of the next BASIC line.

By the way, these routines were designed for use in PCs having an 8K memory module installed. If you have an unexpanded unit or a 4K module, then substitute the value &48C5 wherever &38C5 appears in the listing.

Program *Op-Aids*

```
 1:ARUN :WAIT 200
   :PRINT STATUS
   1;" BYTES NOW
   RESIDENT"
 2:N=(STATUS 2)-1
   :SF=1:GOSUB 12
 3:WAIT :PRINT "P
   ROGRAMS FROM "
   ;"38C5 TO ";B$
   :END
10:"J"AREAD N:SF=
   0
12:A$="0123456789
   ABCDEF":B$=""
15:FOR X=1TO 4
20:Q=INT (N/16)
30:R=N-16*Q
40:B$=MID$ (A$,R+
   1,1)+B$:N=Q
45:NEXT X:IF SF
   THEN RETURN
50:WAIT :PRINT B$
   :END
200:"K"INPUT "LINE
    NUMBER?",N
210:X=&38C5
220:IF 256*(PEEK X
   )+PEEK (X+1)=N
   PRINT X:END
230:X=X+3+PEEK (X+
   2):GOTO 220
```


The POCKET COMPUTER NEWSLETTER is Available!

By Subscription Only: for a calendar year period (January – December). You get all issues published to date for the calendar year in which you subscribe, at the time you subscribe.

MC/VISA Phone Subscriptions: (203) 888-1946

☐ 1982 Regular Subscriber (Issues 11 – 20). $30.00 in U.S. (U.S. $36.00 to Canada. U.S. $45.00 elsewhere.)

☐ 1982/83 Subscriber (Issues 11 – 30). $60.00 in U.S. (U.S. $72.00 to Canada. U.S. $90.00 elsewhere.)

☐ 1983 Regular Subscriber (Issues 21 – 30). $36.00 in U.S. (U.S. $42.00 to Canada. U.S. $50.00 elsewhere.)

Orders must be accompanied by payment in full. We do not issue invoices for the POCKET COMPUTER NEWSLETTER.

Thank you for your remittance.

Name:_____

Addr: _____

City: _____ State: ___ Zip: _____

MC/VISA #: _____ Expires: _____

Signature: _____

P.O. Box 232, Seymour, CT 06483

FROM THE WATCH POCKET

My sources indicate that Hewlett-Packard is running a bit behind in getting the new HP-75 into the hands of users. The original announced date for initial deliveries was September 15th. One exasperated dealer, unable to get a single unit to date, while being circled by cash-in-hand buyers, quipped that, when it came to initial deliveries, HP stood for "Have Patience." So be it. At least they have a cadre of interested buyers standing around waiting for it!

Seems HP isn't the only firm running a little behind on anticipated delivery dates. I have yet to hear of any Sharp ROM modules being delivered in the U.S. I have also heard that the RS-232 interface may be slipped to December.

It also seems that Sharp does not plan to market some of its accessories for the PC-1500 in the U.S. At least, not at the present time. This goes, apparently, for the CE-153 Software Board (a matrix of 140 switches that can be used as an input device). I have also heard that Sharp now has a video display unit available for the PC-1500 in Japan, but has no plans for introducing such a product to the U.S. market.

There is little talk these days of a disk or other high speed mass memory device being planned for the PC-1500. Earlier, I had reports that such units were being studied. It seems to me though, that such devices should be possible. Maybe some independents will take a look at the potential for such accessories. I am, frankly, surprised that no one is yet producing any kind of peripheral device for the unit. Is this because Sharp has been very closed-mouthed about technical information?

The new on-line service being provided by Panasonic on the CompuServe network seems to be worthwhile and timely. I plan to keep an eye on it for awhile and will let you know how it develops. Remember, anyone on CompuServe can access this facility. There are programs on the bulletin that can be adapted to PCs other than Panasonic's HHCs.

Speaking of networks: *PCN* is on the Source (ST4322) as well as CompuServe (74445,1123). I currently check for mail on those networks a few times a week.

There is an outfit called **Elek-Tek, Inc.**, *6557 N. Lincoln Avenue, Chicago, IL 60645*, phone (800) 621-1269, advertising the PC-1211 for $94 and the CE-122 Printer/Cassette interface for $64! They also carry the PC-1500 as well as Hewlett-Packard equipment. You might want to check 'em out if you are in the market. Also, I continue to receive favorable reports on the speedy service and low prices provided by: **Tam's, Inc.**, *14932 Garfield Avenue, Paramount, CA 90723*, with an order phone number of (800) 421-5188, as well as by **Atlantic NE Marketing**, *PO Box 291, Marblehead, MA 01945*, phone (617) 639-0285.

By the way, if anyone is wondering what kind of program submission to *PCN* would really catch my eye, let me tell you: There is a very serious need at this point in the development of machine-language capabilities on the PC-1500/PC-2 for at least a rudimentary Monitor program. By that, I mean a program, using hexadecimal notation, that will enable a m.l. programmer to perform such basic tasks as examine specific memory locations, alter their contents if desired, perform block memory-to-memory transfers, allow at least the examination of the CPU registers (and possibly the alteration of some), permit the establishment of "breakpoints" in memory, and so forth. While such a program could readily be done in BASIC, a version of such a utility tool done in m.l. has obvious appeal. Everything that is needed to produce such a practical tool has been published in *PCN* during the past few months (particularly in the Special Issue). The existence of such a Monitor program would make it much easier for others to create hybrid or pure machine-language programs. It is virtually a necessity for going on to produce and check out a comprehensive assembler.

Of course, any practical embellishments you might want to add, such as decimal-to-hexadecimal conversion, would be "frosting on the cake" provided they were memory-efficient, etc. I would tend to think that a nice little Monitor with a few bells and whistles could be packed into 1.5 to 2K of memory.

I am pleased to report that a number of people are making progress finding key ROM entry points. Stay tuned! – *Nat Wadsworth, Editor*